OIPE JC182
AUG 27 2001
PATENT & TRADEMARK OFFICE

RS
#4
11-2-01

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

RECEIVED
AUG 29 2001
Technology Center 2100

In re Application of :	RENAUD, Pierre
Serial No.:	09/811,528
Group:	2184
Confirmation No.:	9427
Filed:	March 20, 2001
Examiner:	Unknown
Title:	METHOD AND SYSTEM FOR MULTI-PROTOCOL CLOCK RECOVERY AND GENERATION
Attorney Ref:	PAT 113-2

RECEIVED
AUG 31 2001
Technology Center 2600

Assistant Commissioner of Patents
and Trademarks
Washington, D.C. 20231
U.S.A.

Dear Sirs:

Please find enclosed herewith for submission the certified copy of Canadian Patent Application No. 2,301,436 on which United States Patent Application No. 09/811,528 claims priority.

Respectfully submitted,
RENAUD, Pierre

August 23, 2001
Date

By: Anne Kinsman
Registration No. 45291
BORDEN LADNER GERVAIS LLP
60 Queen Street
Ottawa, ON K1P 5Y7
Telephone 613-237-5160
Facsimile 613-787-3558
E-mail ipott@blgcanada.com

ALK/cdg

Encl.
1. Certified Copy CA Appln. No. 2,301,436

10

Office de la propriété intellectuelle du Canada

Un organisme d'Industrie Canada

Canadian Intellectual Property Office

An Agency of Industry Canada

Bureau canadien des brevets

Certification

Canadian Patent Office

Certification

La présente atteste que les documents ci-joints, dont la liste figure ci-dessous, sont des copies authentiques des documents déposés au Bureau des brevets.

This is to certify that the documents attached hereto and identified below are true copies of the documents on file in the Patent Office.

Specification and Drawings, as originally filed, with Application for Patent Serial No: **2,301,436**, on March 20, 2000, by **SPACEBRIDGE NETWORKS CORPORATION,** assignee of Peter Renaud, for "Method and System for Multi-Protocol Clock Recovery and Generation".

S. Grégoire

Agent certificateur/Certifying Officer

June 26, 2001

Date

Canada

(CIPO 68)
01-12-00

OPIC CIPO

ABSTRACT

A system and method for multi-protocol clock recovery and generation in a broadband digital wireless modem. The system includes a time stamp extractor for extracting an embedded timestamp from an incoming data stream, and a clock controller for providing a clock reference signal based on the extracted timestamp.

METHOD AND SYSTEM FOR MULTI-PROTOCOL CLOCK RECOVERY AND GENERATION

FIELD OF THE INVENTION

The present invention relates to a method and system for clock recovery and generation in a multi-protocol environment. In particular, the present invention relates to a method and system for clock recovery and generation in a multi-protocol broadband wireless digital modem architecture.

BACKGROUND OF THE INVENTION

In the field of wireless modem clock recovery, a problem exists for digitally performing clock synchronisation across different link layer protocols. Attempts to solve this problem have so for been directed to the use of mixed digital/analogue circuitry. Such solutions can generally be classified into the following two categories: (a) digital to analogue converters and VCOs, and (b) direct digital synthesis. The present invention relates to the latter category.

An exemplary prior art system that utilises the ADC/VCO category to perform the frequency synthesis function is disclosed by US Patent 5,699,392 entitled: "Method and system for the recovery of an encoder clock from an MPEG-2 transport stream." Although the system described under this US patent adequately performs the intended function of frequency synthesise, its cost and complexity is high compared to an all digital implementation that can otherwise can be integrated into an ASIC. Furthermore, US Patent 5,699,392 is tailored to work with the ITU H.222 protocol for a 42 bit PCR and may not be well suited to the DOCSIS protocol applications that have a time stamp of 32 bits.

Prior art references related to the present invention include: US Patent 5,699,392, "Method and system for the recovery of an encoder clock from an MPEG-2 transport stream"; US Patent 5,287,182, " Timing recovery for variable bit-rate video on asynchronous transfer mode (ATM) networks"; and US Patent 5,007,070, " Service clock recovery circuit"

In view of the limitations in the cited prior art, there is clearly desirable to provide an economical method and system for multi-protocol clock recovery and generation system that will allow for clock recovery and synchronisation to be protocol-programmable and fully digital.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a programmable multi-protocol clock recovery and generation system and method.

In a first aspect, the present invention provides a clock recovery and generation system for a digital modem, comprising:

a time stamp extractor for extracting an embedded timestamp from an incoming data stream

a clock controller for providing a clock reference signal based on the extracted timestamp.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the present invention will now be described, by way of example only, with reference to the attached Figures, wherein:

Figure 1 is a block diagram of the system according to the present invention;

Figure 2 is a block diagram of the DDS according to the present invention;

Figure 3 shows waveforms for use in the DDS according to the present invention;

Figure 4 shows a resulting waveform sequence according to the present invention;

Figure 5 shows a lookup table for one frequency setting;

Figure 6 is a block diagram of a timestamp extractor according to the present invention;

Figure 7 is a block diagram of a clock controller according to the present invention; and

Figure 8 is a circuit diagram for handoff from the timestamp extractor to the clock controller according to the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The system of the present invention, described herein as the clock_rg circuit, is shown in Fig. 1, and is constructed from two major blocks: the timestamp extractor and the clock controller. The block has two modes of operation: basestation and terminal. In the basestation mode, the circuit generates a local reference clock and a timestamp for the terminals. In the terminal mode, the circuit recovers the timestamp from the data stream,

which it uses to synchronize a local DDS clock.

The timestamp extractor circuit parses the incoming data stream and extracts the embedded timestamp. The circuit is based on a programmable byte processor. The user program the circuit with byte manipulation instructions that extract the timestamp depending on the data stream protocol ex: SES-Astra or DOCSIS.

The circuit is activated by the frame sync flag, which indicates the start of an extraction program. The circuit then runs until the end of its instruction sequence, where it halts until the next frame sync flag. The timestamp extracted can be up to 6 bytes long.

To accommodate DOCSIS, a CRC-CCITT 16 and a CRC-CCITT 32 circuit are incorporated into the Timestamp Extractor block which treats parallel data.

The timestamp extractor is sequenced by the byte arrival times. Since there is no guaranteed minimum gap between successive bytes, it is therefore required to perform all parsing operations to be executed in 1 clock cycle. To achieve this, the timestamp extractor performs all branch, store and CRC operations in parallel. The defined instruction word is 64 bits wide and is stored in a register space to provide 32 instructions. The SES-Astra program requires 11 instructions and DOCSIS, 20 instructions.

Once a timestamp is recovered from the incoming data stream a flag is set and the timestamp is presented to the Clock Control block.

The clock controller operates in to modes: basestation or terminal. In the basestation mode it is programmed to generate a reference clock and from the same clock, a timestamp. In the terminal mode, the circuit is programmed to synchronize a local DDS clock to the basestation reference clock. The synchronization method is based on comparing the local timestamp with the one sent by the basestation. The clock control block also generates the superframe pulse.

The HCPU is the control interface block, which is slaved to the ASIC's 860/8260 CPU interface block.

The HREG block contains the registers shared between the HCPU for programming and the engine blocks for operation control. It is detached from the HCPU because it is part of the respective engine's clock domain.

The TimeStamp Extractor block is activated for the terminal mode. Its main function is to recover the protocol dependent timestamp from the downstream data. The block

is configured for a particular protocol operation via the HCPU register settings. After initialization, the block executes the programmed extraction algorithm block every time the frame sync is activated. If a timestamp is found, its value is loaded into the TS register and the block then idles until the next frame sync.

The DDS Clock is a direct digital synthesis clock circuit. This engine runs off a high-speed clock reference in order to synthesize a lower but variable frequency.

The Clock Control is the main control engine. This block performs the following functions. Implements a protocol dependent timing recovery algorithm in the terminal station mode. Provides an accurate time base for base station mode. Locks on and track to an upstream clock reference. Controls the frequency selection of the DDS. Controls the resetting of the DDS TS counter. Filters out erroneous timestamps. Monitors the timestamp arrival times. Synchronizes the timestamp values, which are generated from 2 asynchronous clocks.

Fig. 2 is a block diagram of the DDS block with signal inputs and outputs, as detailed in the following list.

Ref_clk:	INPUT	Clock that operates the FSM and is used as the reference for the synthesized output clock.
dds_en:	INPUT.	DDS FSM enable; reset and operation control.
dds_load:	INPUT	DDS frequencey parameter load control. When asserted, DDS will wait until current clock cycle is complete then will restart FSM with new frequency parameters.
dds_clk:	OUTPUT.	Synthesized clock output
dds_tb_flg:	OUTPUT.	DDS timebase flag. Asserted when all parameters have been counted down to zero.
dds_en:	INPUT.	FSM enable; reset and operation control.
Sl_cnt:	INPUT.	Sub loop count frequency parameter.
Sl_np_cnt:	INPUT.	Sub loop nominal period count frequency parameter.
Sl_wf_type:	INPUT.	Sub loop waveform type frequency parameter.
end_np_cnt:	INPUT.	Timebase end nominal period count frequency parameter.

To construct a nominal 27mhz with a 50% duty cycle clock from a higher system clock requires a binary multiple: 54, 108, 216, etc. For physical ASIC considerations and for resolution requirements, the 108mhz frequency is chosen (assuming that the duty cycle variation is acceptable). The types of possible waveforms are shown in Fig. 3.

By combining a sequence of nominal with either a short or long waveform, the DDS can generate any frequency that varies in steps. For example, a sequence of 269999982 NP (nominal period) interspersed with 24 SP (short period) gives an average frequency of 2700000.6 Hz over a duration of 10 sec. The resulting waveform sequence is shown in Fig. 4.

The DDS block is implemented with a look up table for the programmable frequency values and a state machine to run the sequence based on a frequency error input value. The lookup table entry for one frequency setting is as shown in Fig. 5.

The TimeStamp Extractor block, shown in Fig. 6, is based on a CISC type processor that executes user programmed instructions to parse through an incoming data stream and extract a timestamp value. The timestamp extractor's features: extracting timestamp values from incoming data stream; qualifying timestamp for clock control block; count timestamp extraction errors for SW diagnostics.

The Instruction Format for the TimeStamp Extractor (TSE) is shown in Appendix C. The TSE operates on 2 main input streams: frame sync and data bytes that arrive at the TSE's clock rate; and RAM instructions.

The TSE is able to process the data on every clock. This constrains the instruction word format to be wide enough to describe all of the different types of operations that occur in parallel as demonstrated by the three pseudo code protocol algorithms described at the end of this document.

Fig. 7 shows the clock controller. The clock controller's features: detecting loss of timestamp for SW diagnostics; sets DDS's frequency under direct SW control, automatically varies DDS's clock frequency according to delta between local and extracted timestamps.

The method of the present invention will now be described. In the SES-Astra clock recovery mode:

SW enables and programs CLOCK_RG for nominal frequency of 27Mhz. Clock ready flag is off; hence TX mod is off.

SW enables RX demod to receive incoming data stream.

SW programs first PID value for timestamp extraction. If PCR PID is unknown, SW can cycle through PID values until the prc_rdy flag is set, indicating that a possible timestamp has been found. PCR value is accessible by SW. Its occurrence is triggered by a TS_RDY interrupt.

CLOCK_RG waits for 1st timestamp to preset internal counter and enable counting. CLOCK_RG uses the 27Mhz to increment the 42 bit counter where first 33 bits are in units of 90Khz (base) and last 9 bits are remainder of 27Mhz divided by 300 (ext).

Upon arrival of 2nd timestamp, this value is compared to timestamp generated via internal clock programmed for 27Mhz. Delta instructs SW application, which range of frequency parameters to load into frequency selection table. Range is determined by on board reference clock uncertainty and specified system clock constraints: LTE 75 * 10**-3. This gives .75 Hz in 10 sec. Therefore a table of 32 values where each one has an accuracy of .1Hz provides 200 sec of automatic clock tracking before SW needs to update the frequency parameter table. The SW then sets CLOCK_RG for tracking mode.

When DDS reaches end of frequency timebase (10 sec) it flags the CLOCK_RG.

The following ts_rdy signal resets the DDS for phase and new frequency table based on last valid PCR timestamp delta. The internal counter is updated with this timestamp. If this PCR is invalid (out of range due to bit error or discontinuity flag set), CLOCK_RG waits for next PCR. This loop repeats until either a valid PCR is received or number of invalid PCRs sets interrupt for loss of sync.

CLOCK_RG asserts the superframe flag every time it receives ts_rdy.

CLOCK_RG in tracking mode compares the internal timestamp with the recovered value. It then performs the following based on the difference:

Base delta = 0 (delta due to base station frequency variation specification)

Ext delta 0 = Frequency lock.

Ext delta 1,2 = Frequency lock. Index to next or second next frequency parameters in table and reset timestamp value. If index not in able, then interrupt SW for new frequency parameter table.

Ext delta > 2 = ignored on first and second occurrence, it is assumed that the PCR is in error, set SW interrupt to record event. On third occurrence, CLOCK_RG operation is reset via SW.

Base delta > 0 = ignored on first and second occurrence, it is assumed that the PCR is in error, interrupt is set for SW to record event. On third occurrence, CLOCK_RG operation is reset via SW.

CLOCK_RG sets SW interrupt and resets clock_rdy when it detects more than 3 consecutive discontinuity indicator flags.

CLOCK_RG sets SW interrupt and resets clock_rdy when it detects a 600msec period with no PCR TS.

In clock generation mode: SW enables and programs CLOCK_RG's DDS for nominal frequency of 27Mhz. Clock ready flag is on.

CLOCK_RG uses the 27Mhz to increment the 42 bit counter where first 33 bits are in units of 90Khz and last 9 bits are remainder of 27Mhz divided by 300.

SW programs and enables the superframe counter to generate the SuperFrame pulse.

In DOCSIS (clock recovery mode): SW enables and programs CLOCK_RG for nominal frequency of 10.24Mhz. Clock ready flag is off; hence TX mod is off.

SW enables RX demod to receive incoming data stream.

SW programs PID (1FFE) value for timestamp extraction. CMTS timestamp value is accessible by SW. Its occurrence is triggered by a TS_RDY interrupt.

CLOCK_RG waits for 1st timestamp to preset internal counter and enable counting.

Upon arrival of 2nd timestamp (200ms later), this value is compared to timestamp generated via internal clock programmed for 10.24Mhz. Delta instructs SW application, which range of frequency parameters to load into frequency selection table. Range is determined by on board reference clock uncertainty and specified system clock constraints: constraints 10ns jitter and 10**-8 drift rate = duration of adjacent 102,400,000 segments within LTE 120ns. This gives 1.2 Hz in 10 sec. Therefore a table of 24 values where each one has an accuracy of .1Hz is required. The SW then sets CLOCK_RG for tracking mode.

When DDS reaches end of frequency timebase (10 sec) it flags the CLOCK_RG. The following ts_rdy signal resets the DDS for phase and new frequency table based on last valid CMTS timestamp delta. The internal counter is updated with this timestamp. If this CMTS is invalid (out of range due to bit error or transport_error_indicator set), CLOCK_RG waits for next

CMTS. This loop repeats until either a valid CMTS is received or number of invalid CMTS sets an interrupt for loss of sync.

CLOCK_RG asserts the superframe flag every time it receives ts_rdy.

CLOCK_RG in tracking mode compares the internal timestamp with the recovered value. It then performs the following based on the difference:

Delta 0 = Frequency lock.

Delta 1,2 = Frequency lock. Index to next or second next frequency parameters in table and reset timestamp value. If index not in table, then interrupt SW for new frequency parameter table.

Delta > 2 = ignored on first and second occurrence, it is assumed that the PCR is in error, set SW interrupt to record event. On third occurrence, CLOCK_RG operation is reset via SW.

CLOCK_RG sets SW interrupt and resets clock_rdy when it detects more than 3 consecutive transport_error_indicator flags.

CLOCK_RG sets SW interrupt and resets clock_rdy when it detects a 600msec period with no CMTS TS.

In clock generation mode: SW enables and programs CLOCK_RG's DDS for nominal frequency of 10.24Mhz. Clock ready flag is on.

CLOCK_RG uses the 10.24Mhz to increment the 32 bit timestamp counter.

Advantages of the present invention include the following features: user programmability for timestamp extraction algorithms, protocol based timestamp generation for base stations, protocol based timestamp extraction for terminal stations, clock recovery for terminal stations, SuperFrame generation and detection, and DDS clock with a resolution of .1Hz

Fig. 8 shows the TS_XTRACTR to CLOCK_CNTL handoff circuit.

The above-described embodiments of the invention are intended to be examples of the present invention. Alterations, modifications and variations may be effected the particular embodiments by those of skill in the art, without departing from the scope of the invention which is defined solely by the claims appended hereto.

SES-Astra Table:

Frequency	SubLoop #	Nominal num	End Period Type	End number	Timebase Frequency
27 Mhz + 10	4	674998	SP	5	100 msec
27 Mhz + 810	324	8332	SP	189	100 msec
27 Mhz + 200	80	33748	SP	100	100 msec
27 Mhz + 100	40	67498	SP	50	100 msec
27 Mhz + .1	4	67499998	SP	5	10 sec
27 Mhz	100	2699999	NP	0	10 sec
27 Mhz - .1	4	67499997	LP	7	10 sec
27 Mhz - 100	40	67498	LP	70	100 msec
27 Mhz -200	80	33748	LP	140	100 msec
27 Mhz - 810	324	8332	LP	27	100 msec
27 Mhz - 1	4	6749997	LP	7	1 sec

Base station accuracy 27Mhz +/- 810hz, rate of change =< 75 * 10 ** -3hz/sec (in 10 sec = .75hz)
Therefore, the expected change in frequency is less than .1 hz.

Table calculations:
1 fsm clock cycle @ 108Mhz = 9.26ns => in 10 sec = 1080000000 fsm cycles
1 PCR clock cycle @ 27Mhz = 37ns => in 10 sec = 270000000 PCR clock cycles

Number of nominal PCR clock cycles = X, number of short PCR clock cycles = Y and long = Z

Equation 1: center frequency

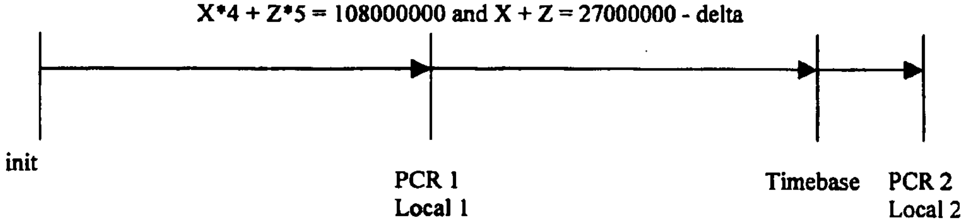
X*4 = 108000000 and X = 27000000 or 10000 sub loops of 2699 NP + 1 NP

Equation 2: > center frequency

X*4 + Y*3 = 108000000 and X + Y = 27000000 + delta

Equation 3: < center frequency

X*4 + Z*5 = 108000000 and X + Z = 27000000 - delta



Error = (local2 –PCR2)/ ((local2 –local1)/ local freq) = delta of counts for time period between PCR1 and PCR2

Approximation: timebase is 10 sec for .1hz accuracy and timebase to PCR2 is usually <.1sec =>
Error ~ (local2 –PCR2)/ 10

DOCSIS Table:

Frequency	SubLoop #	Nominal num	End Period Type	End number	Timebase frequency
10.24 Mhz + 51.2	2048	49998	SP	2560	10 sec
10.24 Mhz + 13.9	556	184170	SP	229	10 sec
10.24 Mhz + 10	40	67498	SP	50	10 sec
10.24 Mhz + .1	4	25599998	SP	5	10 sec
10.24 Mhz	100	10239	NP	0	100 msec
10.24 Mhz - .1	4	25599997	LP	7	10 sec
10.24 Mhz - 1	4	2559997	LP	7	1 sec
10.24 Mhz - 10	4	255997	LP	7	100 msec
10.24 Mhz - 51.2	2048	49997	LP	1536	10 sec

Table calculations:
Base station accuracy +/- 5 ppm = 10.24 Mhz +/- 51.2 Hz. Drift rate =< 10 **-8 per second (10 nsec per sec)
Therefore, the expected change in frequency is equal to .1 hz.

1 fsm clock cycle @ 40.96Mhz = 24.4ns => in 10 sec = 409600000 fsm cycles
1 DDS clock cycle @ 10.24Mhz = 97.66ns => in 10 sec = 102400000 dds clock cycles

Number of nominal period clock cycles = X, number of short period clock cycles = Y and long = Z

Equation 1: center frequency

X*4 = 409600000 and X = 102400000 or 10000 sub loops of 102399 NP + 1 NP

Equation 2: > center frequency

X*4 + Y*3 = 409600000 and X + Y = 102400000 + delta

Equation 3: < center frequency

X*4 + Z*5 = 409600000 and X + Z = 102400000 - delta

HCS control instructions 63:60	RSVD 59	Data control instructions 58:40	RSVD 39: 35	Instruction branch control 34:0

Instruction branch control					
34:31	30:28	27:20	19:12	11:6	5:0
Instruction opcode	Source Data select	Mask operand	Compare operand	True case: 6 bit Ram branch taddr	False case: 6 bit Ram branch faddr

Branch control instruction list:

0: BNOP:	wait for frame sync to restart	
1: JMP:		goto [taddr]
2: JEQ:	if { ([data] AND [mask op]) = [comp op]}	then goto [taddr] else goto [faddr]
3; JGT:	if { ([data] AND [mask op]) > [comp op]}	then goto [taddr] else goto [faddr]
4: JLT:	if { ([data] AND [mask op]) < [comp op]}	then goto [taddr] else goto [faddr]
5: JDBE:	if { ([dbyte] AND [mask op]) = [comp op]}	then goto [taddr] else dec [dbyte] and goto [faddr]
6: JDWE:	if { [dword] = [mask op] &[comp op]}	then goto [taddr] else dec [dword] and goto[faddr]
7: JHCS0EQ:	if { ([data] XOR [mask op]) = HCS[7:0]}	then goto [taddr] else goto [faddr]
8: JHCS1EQ:	if { ([data] XOR [mask op]) = HCS[15:8]}	then goto [taddr] else goto [faddr]
9: JMCS0EQ:	if { ([data] XOR [mask op]) = CRC[7:0]}	then goto [taddr] else goto [faddr]
a: JMCS1EQ:	if { ([data] XOR [mask op]) = CRC[15:8]}	then goto [taddr] else goto [faddr]
b: JMCS2EQ:	if { ([data] XOR [mask op]) = CRC[23:16]}	then goto [taddr] else goto [faddr]
c: JMCS3EQ:	if { ([data] XOR [mask op]) = CRC[31:24]}	then goto [taddr] else goto [faddr]

JEQ, JGT, LPB, JHSCEQ: source data select	
000	Input byte
001	Byte_reg_1
010	Byte_reg_2
011	Byte_reg_3
100	Byte_reg_4
101	DB_reg_0
110	DW_reg_1
111	DW_reg_2

Data control 17:0			
18:16 (58:56)	15:13 (55:53)	12:5 (52:45)	4:0 (44:40)
Opcode	Source Data select	Mask/data operand	Destination Data

Instruction list:
1- 000: DNOP: no operation
2- 001: DSMD: ([source data] AND [mask op]) => [dest. data]
3- 010: DIMD: [mask op] => [dest. data]
4- 011: DTTSRDY: if true flag set [mask op(0)] => TS_RDY
5- 100: DREGADD: ([source data] + [mask op]) => [dest. data]
6- 101: DREGSUB: ([source data] - [mask op]) => [dest. data]
7- 110: DDWINC: DW + 1 => DW
8- 111: DDWDEC: DW - 1 => DW

STOR source data select decode	
000	Input byte
001	Byte_reg_1
010	Byte_reg_2
011	Byte_reg_3
100	Byte_reg_4
101	DB_reg
110	DW_reg_L
111	DW_reg_M

Data Inst Destination	Data decode
00000	X
00001	Byte_reg_1
00010	Byte_reg_2
00011	Byte_reg_3
00100	Byte_reg_4
00101	DB_reg
00110	DW_reg_L
00111	DW_reg_M
01000	HCS_SEED0
01001	HCS_SEED1
01010	MCS_SEED0
01011	MCS _SEED1
01100	MCS _SEED2
01101	MCS _SEED3
01110	TS_RDY
01111	rsvd
10000	TS_reg_1
10001	TS_reg_2
10010	TS_reg_3
10011	TS_reg_4
10100	TS_reg_5
10101	TS_reg_6

CRC control
3:0 (63:60)
Instruction opcode

Instruction list:
1- 0000: CNOP:
2- 0001: LDHCS: [Input byte] => HCS
3- 0010: TFHCS: if true flag set [Input byte] => HCS
4- 0011: FFHCS: if false flag set [Input byte] => HCS
5- 0100: LDMCS: [Input byte] => MCS
6- 0101:TFMCS: if true flag set [Input byte] => MCS
7- 0110:FFMCS: if false flag set [Input byte] => MCS
8- 0111: RHCS: 0000 => HCS
9- 1000: PHCS: FFFF => HCS
10- 1001: RMCS: 00000000 => MCS
11- 1010: PMCS: FFFFFFF => MCS
12- 1011: RHMCS: RHCS and RMCS
13- 1100: PHMCS: PHCS and PMCS

SES ASTRA:

SES Astra or ITU-T H.222.0 timesamp extraction algorithm: (instructions are not executed until byte_rdy occurs). Timestamp arrives every 100 msec or less. Loss of timestamp for *?* msec causes TX shutdown.

```
Frame sync flag→Reset FSM and registers.  Goto step1      /* flag qualifies MPEG SYNC = 47 */

1-  if byte( 7) = 0 TEI and byte(4:0) = PID msb then
        goto step 2                                         /* PID msb match */
    else
        Goto idle state and wait for frame start

2-  if byte(7:0) = PID lsb then
        goto step 3                                         /* PID lsb match */
    else
        Goto idle state and wait for frame start

3- if byte(5) = 1 AFC lsb then                              /* mask in AFC bit */
        goto step 4                                         /* AFC match */
    else
        Goto idle state and wait for frame start

4- If byte > 6 then                                         /* check adaption field length for PCR */
        Goto step 5                                         /* PCR minimum length ok */
    Else
        Goto idle state and wait for frame start

5- If byte(7) = 0 and byte(5) = 1 then                      /* discontinuity error not set and PCR ok */
        Goto step 6
    Else
        Goto idle state and wait for frame start            /* PCR discontinuity or no PCR */

6- Store PCR(47:40)
7- Store PCR(39:32)
8- Store PCR(31:24)
9- Store PCR(23:16)
10- Store PCR(15:8)
11- Store PCR(7:0) and set timestamp flag + Goto idle state and wait for frame start
```

DOCSIS protocol:

Docsis timesamp extraction algorithm: (instructions are not executed until byte_rdy occurs)
Timestamp arrives every 200 msec. Loss of timestamp for 600 msec causes TX shutdown.

```
Frame sync flag→Reset FSM and registers.  Goto step1        /* flag qualifies MPEG SYNC = 47 */

1-  if TEI bit 7 in byte = 0 and PUSI bit 6 in byte = 1 and byte(4:0) = PID msb then
        goto step 2                                         /* PID msb match */
    else
        Goto idle state and wait for frame start

2-  if byte(7:0) = PID lsb then
        goto step 3                                         /* PID lsb match */
    else
        Goto idle state and wait for frame start
3- goto step 4                                              /* ignore AFC */

4- If byte = 0 then                                         /* check pointer field */
        Goto step 5
   Else
        goto step 6

   store byte in Dbyte                                      /* save pointer_field in decrement byte reg*/
   Preset HCS and MCS

5- If byte = ff then                                        /* if stuff bytes then filter */
        goto 5 (set true flag)                              /* loop while ff */
   Else
        Goto 7 (set false flag)

   If false flag set then
        Enable HCS on byte

   Store byte in TempReg (FC byte)

6- If DByte = 0 then                                        /* pointer_field = MAC tail filter */
        goto 5
   Else
        Decrement Dbyte: goto 6                             /* loop until end of MAC tail */

7-  If TempReg (7:6) = Mac specific Header and TempReg (5:1) = Timing Header and TempReg (0) = 0 then
        goto step 8                                         /* Timing MAC header match */
    Else
        Goto step 42                                        /* non Timing MAC header filter */

    Enable HCS on byte                                      /* perform HCS on Mac_parm */
```

```
8-  Store byte in TempWord(15:8) (length msb)
    Enable HCS on byte

9-  Store byte in TempWord(7:0) (length lsb)
    Enable HCS on byte

10- if byte = !HCS msb then
        goto step 11                                   /* HCS msb match */
   else
    Goto idle state and wait for frame start           /* bad CRC wait for next frame */

11- if byte = !HCS lsb then
        goto step 12                                   /* HCS msb match */
   else
    Goto idle state and wait for frame start           /* bad CRC wait for next frame */

   Preset HCS

12- Enable MCS on byte; goto next step                 /* MCS on DA(23:16) */
13- Enable MCS on byte; goto next step                 /* MCS on DA(15:8) */
14- Enable MCS on byte; goto next step                 /* MCS on DA(7:0) */
15- Enable MCS on byte; goto next step                 /* MCS on SA(23:16) */
16- Enable MCS on byte; goto next step                 /* MCS on SA(15:8) */
17- Enable MCS on byte; goto next step                 /* MCS on SA(7:0) */
18- Enable MCS on byte; goto next step                 /* MCS on LEN msb */
    Store byte in DWord                                /* keep length in case wrong Mac msg */
19- Enable MCS on byte; goto next step                 /* MCS on LEN lsb */
    Store byte in Dword                                /* keep length in case wrong Mac msg */
20- Enable MCS on byte; goto next step                 /* MCS on DSAP */
    Decrement DWord
21- Enable MCS on byte; goto next step                 /* MCS on SSAP */
   Decrement DWord
22- Enable MCS on byte; goto next step                 /* MCS on SSAP */
   Decrement DWord
23- Enable MCS on byte; goto next step                 /* MCS on Control */
    Decrement DWord

24- if byte = 1 then                                   /* check version */
       goto 25
   else
       goto 37
    Decrement DWord
    Enable MCS on byte

25- if byte = 1 then                                   /* check type */
       goto 26
   else
       goto 37
    Decrement DWord
   Enable MCS on byte

26-Enable MCS on byte                                  /* MCS on RSVD */
   Decrement DWord
```

```
27- if Dword = 9 then                              /* check last byte before CMTS */
       goto 28
    else
        goto 27                                     /* loop until end */

    Decrement DWord
   Enable MCS on byte

28- Store CMTS(31:24)
     Enable MCS on byte

29- Store CMTS(23:16)
     Enable MCS on byte

30- Store CMTS(15:8)
     Enable MCS on byte

31- Store CMTS(7:0)
     Enable MCS on byte

32- if byte = !MCS(31:24)  then                          /* check CRC */
        goto 33
    else
     Goto idle state and wait for frame start      /* bad MCRC wait for next frame */

33- if byte = !MCS(23:16)  then                          /* check CRC */
        goto 34
    else
     Goto idle state and wait for frame start      /* bad MCRC wait for next frame */

34- if byte = !MCS(15:8)  then                           /* check CRC */
        goto 35
    else
     Goto idle state and wait for frame start      /* bad MCRC wait for next frame */

35- if byte = !MCS(7:0)  then                            /* check CRC */
      Goto idle state and wait for frame start
    else
     Goto idle state and wait for frame start      /* bad MCRC wait for next frame */

    If true flag set then
      Set TS_RDY
```

```
36- if Dword = 5 then                              /* check last byte before CRC */
       goto 37
    else
       goto 36                                     /* loop until end */

    Decrement DWord
    Enable MCS on byte

38- if byte = !MCS(31:24) then                     /* check CRC */
       goto 39
    else
       Goto idle state and wait for frame start    /* bad MCRC wait for next frame */

39- if byte = !MCS(23:16) then                     /* check CRC */
       goto 40
    else
       Goto idle state and wait for frame start    /* bad MCRC wait for next frame */

40- if byte = !MCS(15:8) then                      /* check CRC */
       goto 41
    else
       Goto idle state and wait for frame start    /* bad MCRC wait for next frame */

41- if byte = !MCS(7:0) then                       /* check CRC */
       goto 4                                      /* go to next mac message */
    else
       Goto idle state and wait for frame start    /* bad MCRC wait for next frame */
       Preset HCS

/* wrong FC message */
42- Store byte in DWord(15:8) (length msb)
    Enable HCS on byte                             /* enable HCS on LEN msb */

43- Store byte in DWord(7:0) (length lsb)
    Enable HCS on byte

44- if byte = !HCS msb then
       goto step 45                                /* HCS msb match */
    else
    Goto idle state and wait for frame start       /* bad CRC wait for next frame */

45- if byte = !HCS lsb then
       goto step 46                                /* HCS msb match */
    else
    Goto idle state and wait for frame start       /* bad CRC wait for next frame */

46- If DWord = 0 then                              /* loop until end of MAC frame */
       Goto step 4                                 /* goto stuff byte check */
    Else
       Decrement TempWord
```

SES-ASTRA timestamp instruction program

Addr	HCS op	Data op	Branch op	taddr	faddr
00-	CNOP	DNOP [X] [X] [X]	JEQ [000] [9f] [pid msb]	[01]	[0c]
01-	CNOP	DNOP [X] [X] [X]	JEQ [000] [ff] [pid lsb]	[02]	[0c]
02-	CNOP	DNOP [X] [X] [X]	JEQ [000] [20] [acf]	[03]	[0c]
03-	CNOP	DNOP [X] [X] [X]	JGT [000] [ff] [07]	[04]	[0c]
04-	CNOP	DNOP [X] [X] [X]	JEQ [000] [90] [10]	[05]	[0c]
05-	CNOP	DSMD [000] [ff] [15]	JMP [X] [X] [X]	[06]	[X]
06-	CNOP	DSMD [000] [ff] [14]	JMP [X] [X] [X]	[07]	[X]
07-	CNOP	DSMD [000] [ff] [13]	JMP [X] [X] [X]	[08]	[X]
08-	CNOP	DSMD [000] [ff] [12]	JMP [X] [X] [X]	[09]	[X]
09-	CNOP	DSMD [000] [ff] [11]	JMP [X] [X] [X]	[0a]	[X]
0a-	CNOP	DSMD [000] [ff] [10]	JMP [X] [X] [X]	[0b]	[X]
0b-	CNOP	DIMD [X] [01] [0e]	JMP [X] [X] [X]	[0c]	[X]
0c-	CNOP	DIMD [X] [00] [0e]	JMP [X] [X] [X]	[0c]	[X]

```
constant astrax_code_0 : X"0000 0001 09f1 a04c";
constant astrax_code_1 : X"0000 0001 0ff6 908c ";
constant astrax_code_2 : X"0000 0001 0202 00cc ";
constant astrax_code_3 : X"0000 0001 8ff0 710c";
constant astrax_code_4 : X"0000 0001 0901 014c";
constant astrax_code_5 : X"011f f500 8000 0180";
constant astrax_code_6 : X"011f f400 8000 01c0";
constant astrax_code_7 : X"011f f300 8000 0200";
constant astrax_code_8 : X"011f f200 8000 0240";
constant astrax_code_9 : X"011f f100 8000 0280";
constant astrax_code_10 : X"011f f000 8000 02c0";
constant astrax_code_11 : X"0200 2e00 8000 0300";
constant astrax_code_12 : X"0200 0e00 8000 0300";
```

Example for extracting 4 extra bytes after timestamp:

```
constant astra_code_0 :   X"0000 0001 09f1 a050";
constant astra_code_1 :   X"0000 0001 0ff6 9090 ";
constant astra_code_2 :   X"0000 0001 0202 00d0 ";
constant astra_code_3 :   X"0000 0001 8ff0 7110";
constant astra_code_4 :   X"0000 0001 0901 0150";
constant astra_code_5 :   X"011f f500 8000 0180";
constant astra_code_6 :   X"011f f400 8000 01c0";
constant astra_code_7 :   X"011f f300 8000 0200";
constant astra_code_8 :   X"011f f200 8000 0240";
constant astra_code_9 :   X"011f f100 8000 0280";
constant astra_code_10 :  X"011f f000 8000 02c0";
constant astra_code_11 :  X"011f e400 8000 0300";
constant astra_code_12 :  X"011f e300 8000 0340";
constant astra_code_13 :  X"011f e200 8000 0380";
constant astra_code_14 :  X"011f e100 8000 03c0";
constant astra_code_15 :  X"0200 2e00 8000 0400";
constant astra_code_16 :  X"0200 0e00 8000 0400";
```

DOCSIS timestamp instruction program

Addr	HCS op	Data op		Branch op		taddr	faddr
00-	CNOP	DNOP	[X] [X] [X]	JEQ	[000] [df] [pid msb]	[01]	[12]
01-	CNOP	DNOP	[X] [X] [X]	JEQ	[000] [ff] [pid lsb]	[02]	[12]
02-	CNOP	DNOP	[X] [X] [X]	JMP	[X] [X] [X]	[03]	[X]
03-	PHMCS	DSMD	[000] [ff] [05]	JEQ	[000] [ff] [00]	[04]	[05]
04-	FFHCS	DSMD	[000] [ff] [01]	JEQ	[000] [ff] [ff]	[04]	[06]
05-	CNOP	DSMD	[000] [ff] [10]	JDBE	[000] [ff] [00]	[04]	[05]
06-	LDHCS	DNOP	[X] [X] [X]	JEQ	[001] [ff] [c0]	[07]	[28]
07-	LDHCS	DNOP	[X] [X] [X]	JMP	[X] [X] [X]	[08]	[X]
08-	LDHCS	DNOP	[X] [X] [X]	JMP	[X] [X] [X]	[09]	[X]
09-	CNOP	DNOP	[X] [X] [X]	JHCS1EQ	[000] [ff] [X]	[0a]	[2e]
0a-	CNOP	DNOP	[X] [X] [X]	JHCS0EQ	[000] [ff] [X]	[0b]	[2e]
0b-	LDMCS	DNOP	[X] [X] [X]	JMP	[X] [X] [X]	[0c]	[X]
0c-	LDMCS	DNOP	[X] [X] [X]	JMP	[X] [X] [X]	[0d]	[X]
0d-	LDMCS	DNOP	[X] [X] [X]	JMP	[X] [X] [X]	[0e]	[X]
0e-	LDMCS	DNOP	[X] [X] [X]	JMP	[X] [X] [X]	[0f]	[X]
0f-	LDMCS	DNOP	[X] [X] [X]	JMP	[X] [X] [X]	[10]	[X]
10-	LDMCS	DNOP	[X] [X] [X]	JMP	[X] [X] [X]	[11]	[X]
11-	LDMCS	DSMD	[000] [ff] [07]	JMP	[X] [X] [X]	[12]	[X]
12-	LDMCS	DSMD	[000] [ff] [06]	JMP	[X] [X] [X]	[13]	[X]
13-	LDMCS	DDWDEC	[X] [X] [X]	JMP	[X] [X] [X]	[14]	[X]
14-	LDMCS	DDWDEC	[X] [X] [X]	JMP	[X] [X] [X]	[15]	[X]
15-	LDMCS	DDWDEC	[X] [X] [X]	JMP	[X] [X] [X]	[16]	[X]
16-	LDMCS	DDWDEC	[X] [X] [X]	JMP	[X] [X] [X]	[17]	[X]
17-	LDMCS	DDWDEC	[X] [X] [X]	JEQ	[000] [ff] [01]	[18]	[28]
18-	LDMCS	DDWDEC	[X] [X] [X]	JEQ	[000] [ff] [01]	[19]	[28]
19-	LDMCS	DDWDEC	[X] [X] [X]	JMP	[X] [X] [X]	[1a]	[X]
1a-	LDMCS	DNOP	[X] [X] [X]	JDWE	[000] [ff] [09]	[1b]	[1a]
1b-	LDMCS	DSMD	[000] [ff] [13]	JMP	[X] [X] [X]	[1c]	[X]
1c-	LDMCS	DSMD	[000] [ff] [12]	JMP	[X] [X] [X]	[1d]	[X]
1d-	LDMCS	DSMD	[000] [ff] [11]	JMP	[X] [X] [X]	[1e]	[X]
1e-	LDMCS	DSMD	[000] [ff] [10]	JMP	[X] [X] [X]	[1f]	[X]
1f-	CNOP	DNOP	[X] [X] [X]	JMCS3EQ	[000] [ff] [X]	[20]	[2e]
20-	CNOP	DNOP	[X] [X] [X]	JMCS2EQ	[000] [ff] [X]	[21]	[2e]
21-	CNOP	DNOP	[X] [X] [X]	JMCS1EQ	[000] [ff] [X]	[22]	[2e]
22-	CNOP	DTTSRDY	[X] [01] [X]	JMCS0EQ	[000] [ff] [X]	[2e]	[2e]
23-	LDMCS	DDWDEC	[X] [X] [X]	JDWE	[000] [ff] [5]	[24]	[23]
24-	CNOP	DNOP	[X] [X] [X]	JMCS3EQ	[000] [ff] [X]	[25]	[2e]
25-	CNOP	DNOP	[X] [X] [X]	JMCS2EQ	[000] [ff] [X]	[26]	[2e]
26-	CNOP	DNOP	[X] [X] [X]	JMCS1EQ	[000] [ff] [X]	[27]	[2e]
27-	CNOP	DNOP	[X] [X] [X]	JMCS0EQ	[000] [ff] [X]	[04]	[2e]
28-	LDHCS	DSMD	[000] [ff] [07]	JMP	[X] [X] [X]	[29]	[X]
29-	LDHCS	DSMD	[000] [ff] [06]	JMP	[X] [X] [X]	[2a]	[X]
2a-	CNOP	DNOP	[X] [X] [X]	JHCS1EQ	[000] [ff] [X]	[2b]	[2e]
2b-	CNOP	DNOP	[X] [X] [X]	JHCS0EQ	[000] [ff] [X]	[2c]	[2e]
2c-	LDMCS	DNOP	[X] [X] [X]	JDWE	[000] [ff] [00]	[04]	[2c]
2d-	CNOP	DNOP	[X] [X] [X]	BNOP	[X] [X] [X]	[X]	[X]

Register List:

Timestamp extractor:

```
#
# timestamp extractor register set
#

DEVICE_ID = 6'd34;
DEVICE_NAME = "cpu_ts_xtrctr";

# block enable register
# bit 0 ts_xtrctr

REGISTER ARRAY 1:
      TITLE = "timestamp extraction control";
      NAME = "control";
      DEFAULT = 1'h0;
      ATTR = R/W;
END;

# timestamp extraction program registers
# 64 locations for DOCSIS
REGISTER ARRAY 64:
      TITLE = "timestamp extraction program instructions";
      NAME = "tsx_code";
      DEFAULT = 64'h0;
      ATTR = R/W;
END;

# SW access points
# status                input to HREG
# bit 0 ts_rdy
# bit 1 HCS error
# bit 2 MCS error

REGISTER:
      TITLE = "ts status bits";
      NAME = "ts_status";
      DEFAULT = 3'h0;
      ATTR = R/E;
END;

# recovered data
REGISTER:
      TITLE = " timestamp data lsb";
      NAME = "ts_reg_4_1";
      DEFAULT = 32'h0;
      ATTR = R/E;
END;
```

```
REGISTER:
      TITLE = " timestamp data msb";
      NAME = "ts_reg_6_5";
      DEFAULT = 16'h0;
      ATTR = R/E;
END;

REGISTER:
      TITLE = " general purpose registers";
      NAME = "gen_byte_4_1";
      DEFAULT = 32'h0;
      ATTR = R/E;
END;
```

Clock recovery and generation:

```
#
# Clock recovery and generation register set
#

DEVICE_ID = 5'd29;
DEVICE_NAME = "cpu_ts_clk_cntl";

# cpu access diag register
REGISTER:
      TITLE = "cpu access ID reg";
      NAME = "blkid";
      DEFAULT = 16'h0;
      ATTR = R/E;
END;

# block enable register
# bit 0 ts_clk_cntl_en  : enable the clock control block
# bit 1 ts_clk_auto_en  : enable auto mode for clock control block
# bit 2 wtb_count_en    : window timebase counter enable
# bit 3 sf_cntl_en      : superframe block enable
# bit 4 sf_mode         : superframe block rx/tx recover or generate
mode
# bit 5 sf_ts_ld        : superframe timestamp load control
# bit 6 ts_type         : timestamp SES/DOCSIS type
# bit 7 dds_en          : enable the DDS block
# bit 8 clock_lock_cntl : force clock lock
# bit 9 swts_load       : load timestamp
# bit 10 swts_inc_load  : load increment timestamp

REGISTER:
      TITLE = "clock recovery and generation control";
      NAME = "control";
      DEFAULT = 11'h0;
      ATTR = R/W;
END;

# status                input to HREG
# bit 0 dds_tb_flg      : DDS timebase flag
# bit 1 ts_loss_flg     : loss of timestamp flag
# bit 2 fp_attn_flg     : frequency parameter table update attention
flag
# bit 3 ts_rdy_reg      : timestamp data and counter ready
# bit 4 ts_delta_err    : timestamp delta range error
# bit 5 fp_tbl_of_err   : frequency parameter table overflow error
# bit 6 fp_tbl_uf_err   : frequency parameter table underflow error
# bit 7 clock_lock      : clock lock indicator

REGISTER:
      TITLE = "clock recovery and generation status";
      NAME = "status";
      DEFAULT = 8'h0;
      ATTR = R/E;
END;
```

```
# recovered timestamp loss threshold: ts_loss_thrshld
REGISTER:
      TITLE = "timestamp loss threshold";
      NAME = "ts_loss_thrshld";
      DEFAULT = 8'h0;
      ATTR = R/W;
END;

# timestamp delta maximum filter
REGISTER:
      TITLE = "timestamp delta max";
      NAME = "ts_delta_max";
      DEFAULT = 32'h0;
      ATTR = R/W;
END;

# window timebase terminal count wtb_tc
REGISTER:
      TITLE = "window timebase terminal count";
      NAME = "wtb_tc";
      DEFAULT = 32'h0;
      ATTR = R/W;
END;

# timestamp load value

REGISTER:
      TITLE = " timestamp load lsb";
      NAME = "swts_field_lsb";
      DEFAULT = 32'h0;
      ATTR = R/W;
END;

REGISTER:
      TITLE = " timestamp load msb";
      NAME = "swts_field_msb";
      DEFAULT = 10'h0;
      ATTR = R/W;
END;

# increment timestamp load value

REGISTER:
      TITLE = " timestamp inc load lsb";
      NAME = "swts_inc_field";
      DEFAULT = 8'h0;
      ATTR = R/W;
END;

# superframe timestamp compare value

REGISTER:
      TITLE = " superframe timestamp lsb";
      NAME = "sf_ts_lsb";
      DEFAULT = 32'h0;
      ATTR = R/W;
END;
```

```
REGISTER:
      TITLE = " superframe timestamp msb";
      NAME = "sf_ts_msb";
      DEFAULT = 10'h0;
      ATTR = R/W;
END;

# sf_tc_cnt         --  superframe terminal count
REGISTER:
      TITLE = " superframe generator terminal count";
      NAME = "sf_tc_cnt";
      DEFAULT = 32'h0;
      ATTR = R/W;
END;

# tsd_field_lsb   :access tsd read data for SW
# tsd_field_msb   :access tsd read data for SW

REGISTER:
      TITLE = " timestamp data lsb";
      NAME = "tsd_field_lsb";
      DEFAULT = 32'h0;
      ATTR = R/E;
END;

REGISTER:
      TITLE = " timestamp data msb";
      NAME = "tsd_field_msb";
      DEFAULT = 10'h0;
      ATTR = R/E;
END;

# tsc_field_lsb   :access local ts counter read data for SW
# tsc_field_msb   :access local ts counter read data for SW

REGISTER:
      TITLE = " timestamp count lsb";
      NAME = "tsc_field_lsb";
      DEFAULT = 32'h0;
      ATTR = R/E;
END;

REGISTER:
      TITLE = " timestamp count msb";
      NAME = "tsc_field_msb";
      DEFAULT = 10'h0;
      ATTR = R/E;
END;

#
# frequency parameters
#

REGISTER ARRAY 16:
      TITLE = "Freq parameter NP number per loop slow ";
      NAME = "fp_lnp_s";
```

```
      DEFAULT = 27'h0;
      ATTR = R/W;
END;

REGISTER ARRAY 16:
      TITLE = "Freq parameter loop number, loop type and NP number for
end sequence slow ";
      NAME = "fp_lte_s";
      DEFAULT = 25'h0;
      ATTR = R/W;
END;

REGISTER:
      TITLE = "Freq parameter NP number per loop nominal ";
      NAME = "fp_lnp_n";
      DEFAULT = 27'h0;
      ATTR = R/W;
END;

# 27 Mhz value = 2932df

REGISTER:
      TITLE = "Freq parameter loop number, loop type and NP number for
end sequence nominal ";
      NAME = "fp_lte_n";
      DEFAULT = 25'h0;
      ATTR = R/W;
END;

# 27 Mhz value = 64

REGISTER ARRAY 15:
      TITLE = "Freq parameter NP number per loop fast ";
      NAME = "fp_lnp_f";
      DEFAULT = 27'h0;
      ATTR = R/W;
END;

REGISTER ARRAY 15:
      TITLE = "Freq parameter loop number, loop type and NP number for
end sequence fast ";
      NAME = "fp_lte_f";
      DEFAULT = 25'h0;
      ATTR = R/W;
END;
```

We claim:

1. A clock recovery and generation system for a digital modem, comprising:

a time stamp extractor for extracting an embedded timestamp from an incoming data stream

a clock controller for providing a clock reference signal based on the extracted timestamp.

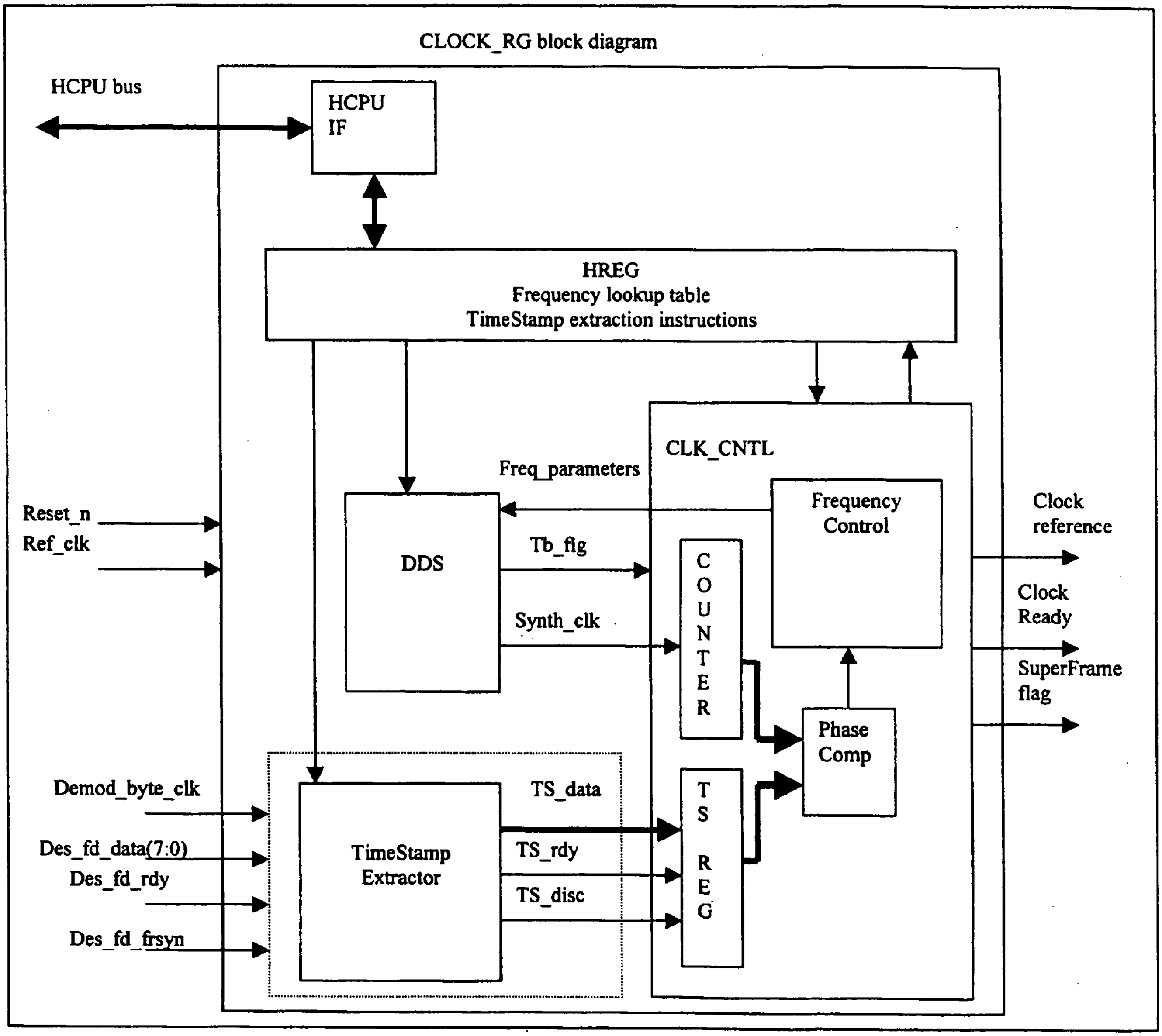

CLOCK_RG block diagram
HCPU bus
HCPU IF
HREG
Frequency lookup table
TimeStamp extraction instructions
CLK_CNTL
Freq_parameters
Frequency Control
Reset_n
Ref_clk
DDS
Tb_flg
Synth_clk
COUNTER
Clock reference
Clock Ready
SuperFrame flag
Phase Comp
Demod_byte_clk
Des_fd_data(7:0)
Des_fd_rdy
Des_fd_frsyn
TimeStamp Extractor
TS_data
TS_rdy
TS_disc
TS REG

Fig. 1

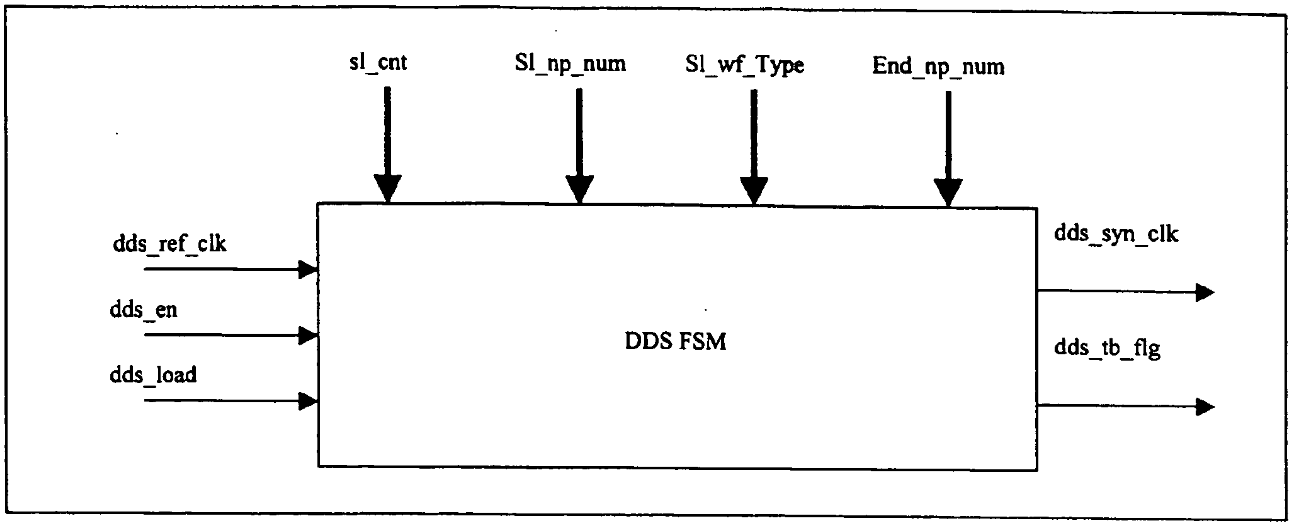

sl_cnt
Sl_np_num
Sl_wf_Type
End_np_num
dds_ref_clk
dds_en
dds_load
DDS FSM
dds_syn_clk
dds_tb_flg

Fig. 2

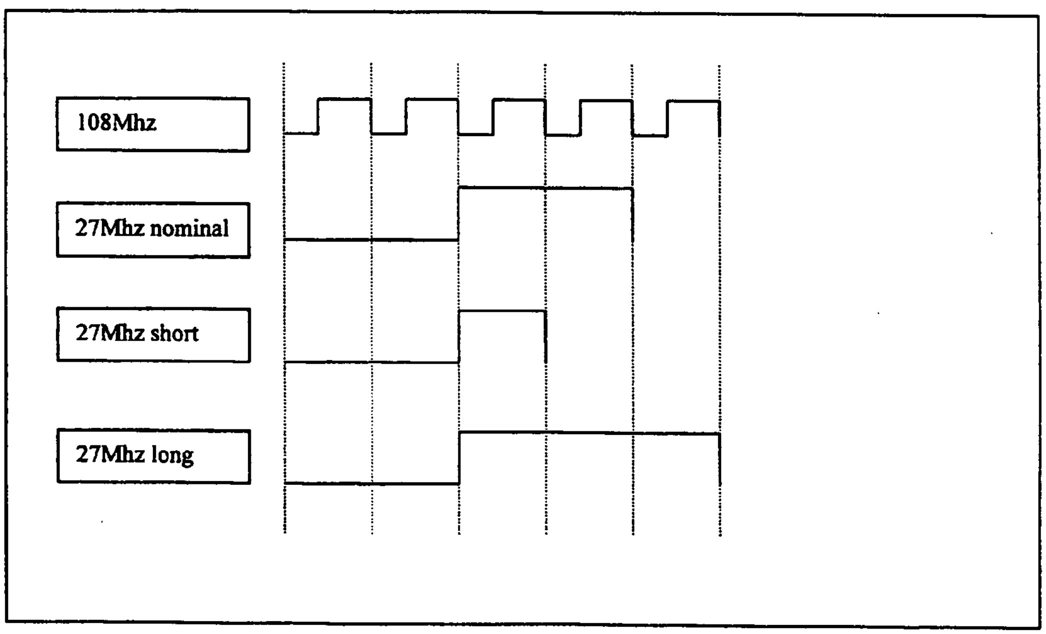

108Mhz
27Mhz nominal
27Mhz short
27Mhz long

Fig. 3

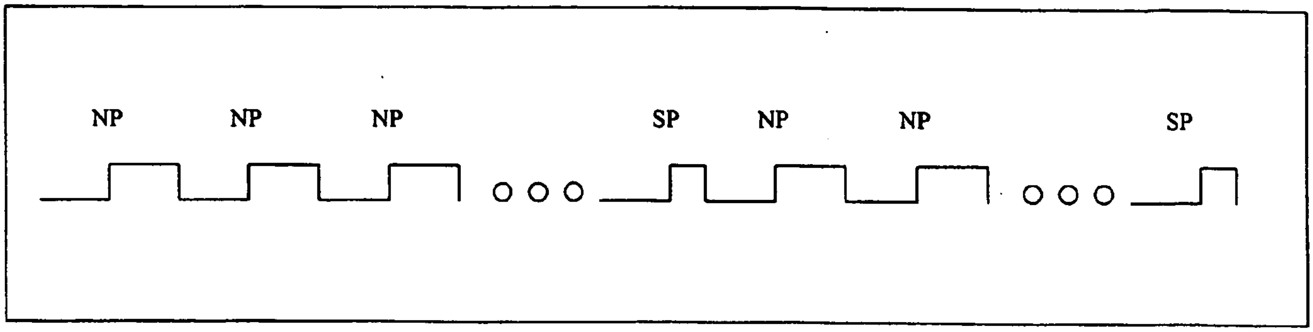


Fig. 4

Fp_lnp_X 26:0
Number of NP in a sub loop

Fp_lte_X 24:13	Fp_lte_X 12:11	Fp_lte_X 10:0
End number of NP to complete 10 sec	End of loop type: NP. SP or LP	Number of repeatable sub loops

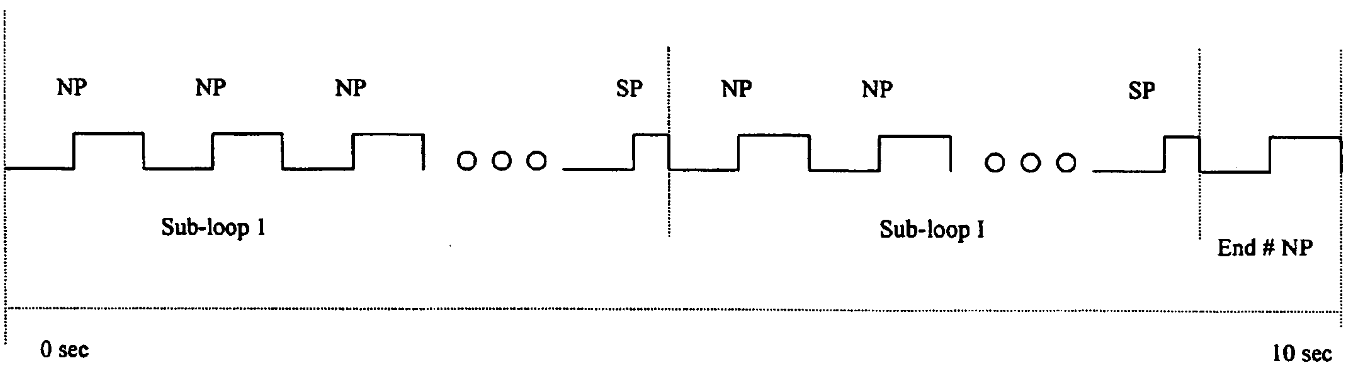


Fig. 5

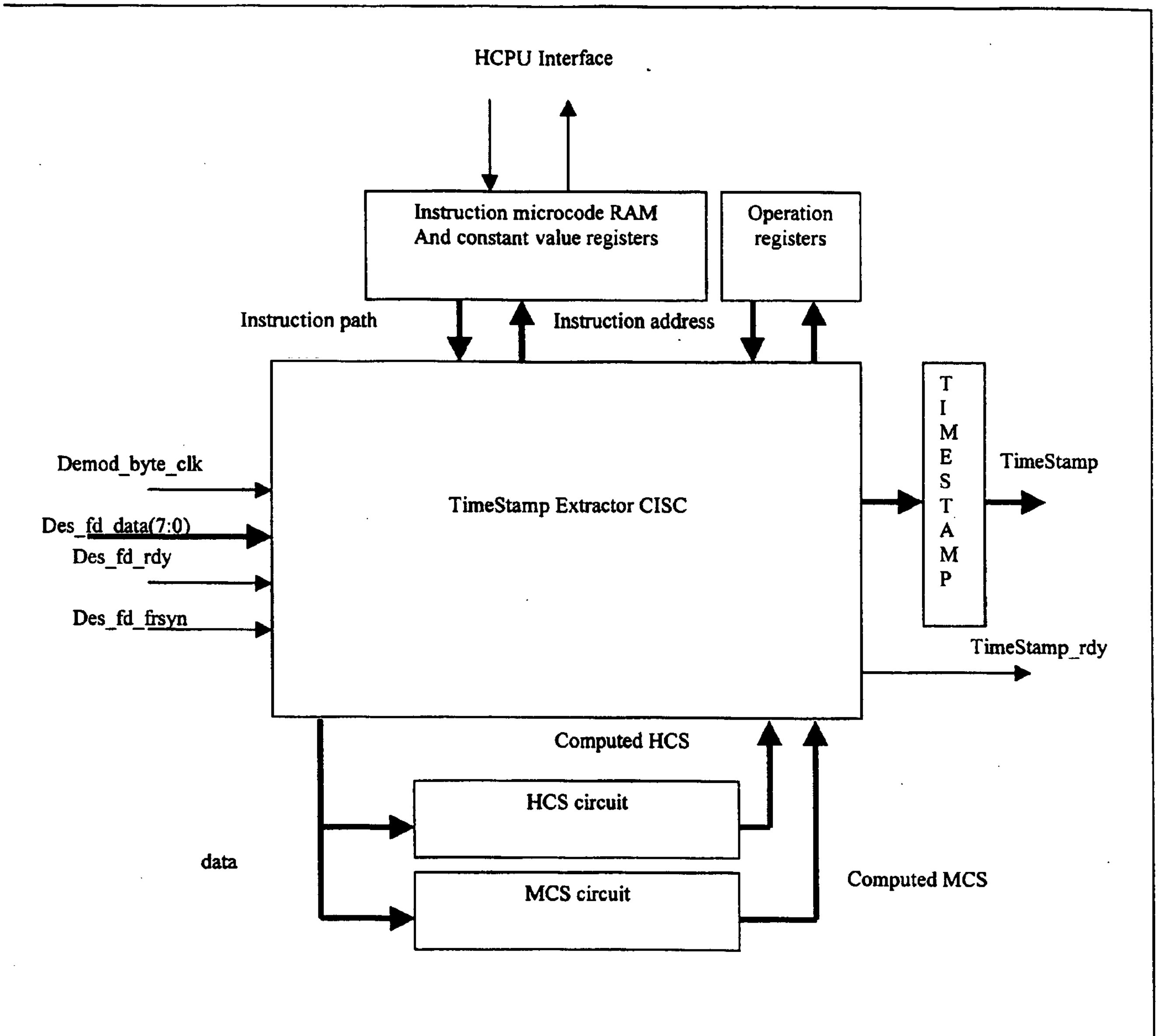

HCPU Interface
Instruction microcode RAM
And constant value registers
Operation
registers
Instruction path
Instruction address
TimeStamp Extractor CISC
Demod_byte_clk
Des_fd_data(7:0)
Des_fd_rdy
Des_fd_frsyn
TIMESTAMP
TimeStamp
TimeStamp_rdy
Computed HCS
HCS circuit
data
MCS circuit
Computed MCS

Fig. 6

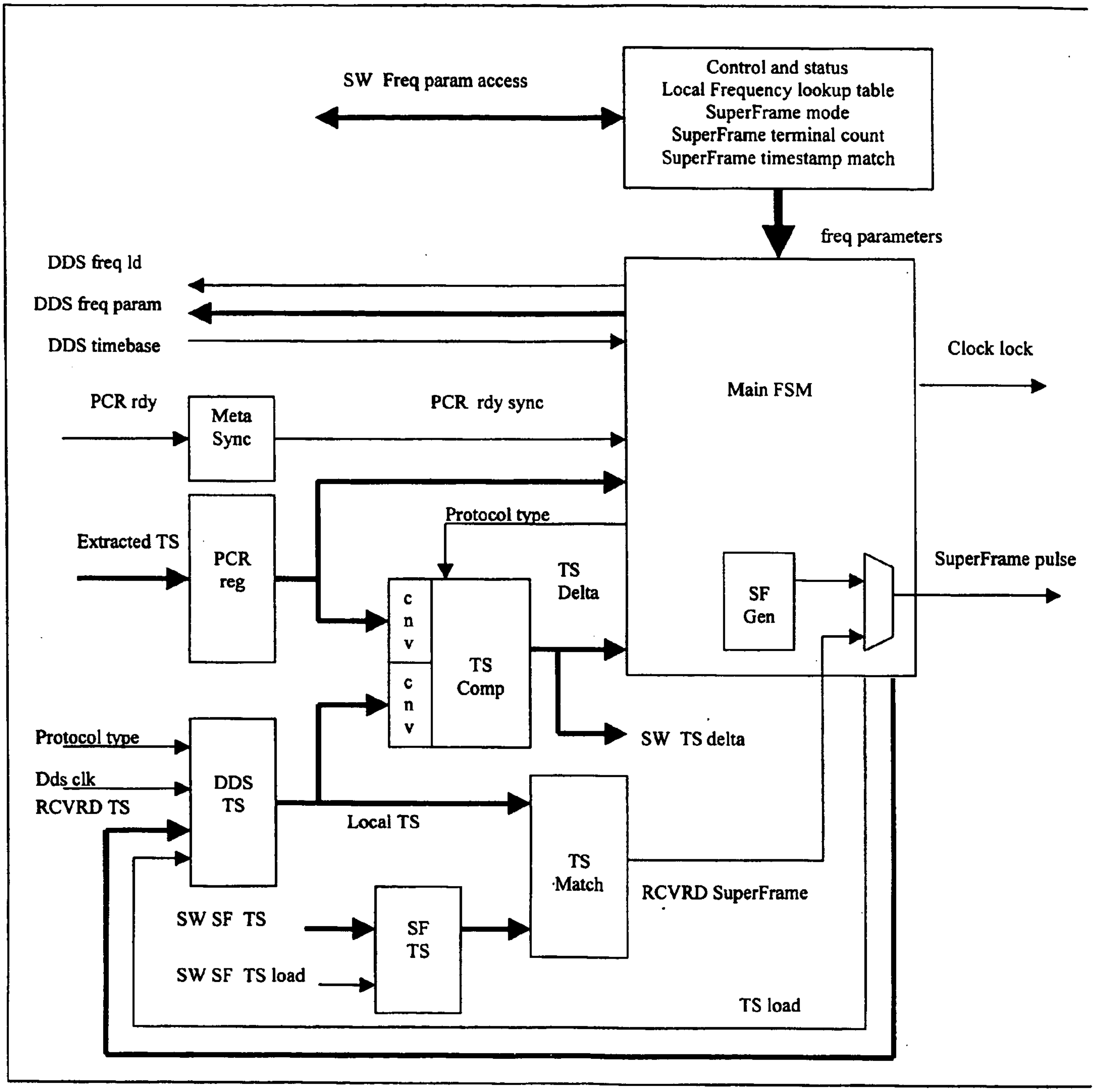


Fig. 7

Borden Ladner Gervais LLP / s.r.l.

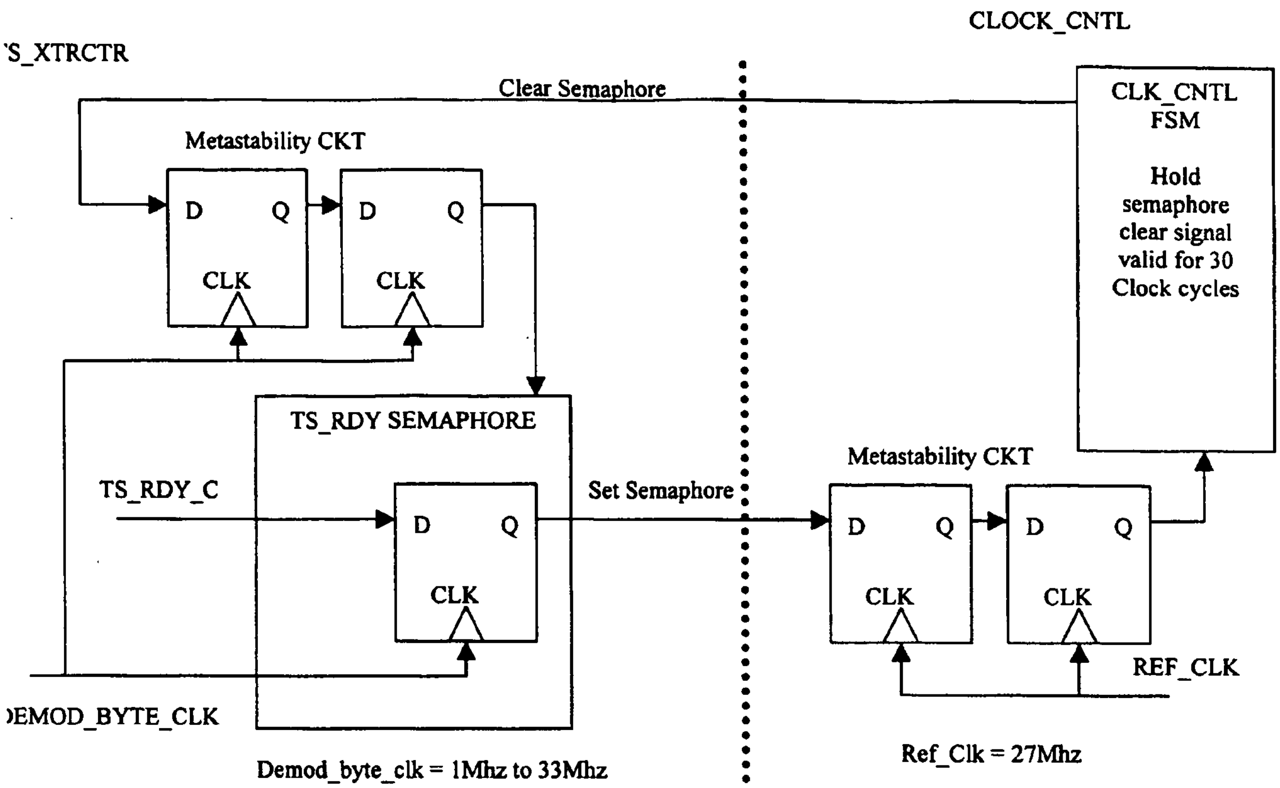

CLOCK_CNTL
'S_XTRCTR
Clear Semaphore
Metastability CKT
D
Q
CLK
D
Q
CLK
CLK_CNTL
FSM
Hold
semaphore
clear signal
valid for 30
Clock cycles
TS_RDY SEMAPHORE
TS_RDY_C
D
Q
CLK
Set Semaphore
Metastability CKT
D
Q
CLK
D
Q
CLK
REF_CLK
)EMOD_BYTE_CLK
Demod_byte_clk = 1Mhz to 33Mhz
Ref_Clk = 27Mhz

Fig. 8